

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE SERVERU UNDERTOW SE SYSTÉMEM JENKINS CI

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE
AUTHOR

Bc. JAKUB BARTEČEK

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

INTEGRACE SERVERU UNDERTOW SE SYSTÉMEM JENKINS CI

INTEGRATION OF JENKINS CI WITH UNDERTOW

SEMESTRÁLNÍ PROJEKT

TERM PROJECT

AUTOR PRÁCE

AUTHOR

Bc. JAKUB BARTEČEK

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. PETR MÜLLER

BRNO 2014

Abstrakt

Tento semestrální projekt se zabývá nahrazením webového serveru v systému Jenkins CI za server Undertow. Server Undertow by měl být potenciálně rychlejší než současné řešení a celkově lepší. V práci jsou popsány obecné informace o komponentách a programech, které se této problematice týkají. Nejdůležitější částí práce je analýza současného stavu a návrh způsobu integrace. Samotná integrace není provedena v rámci semestrálního projektu, ale je předmětem navazujícího diplomového projektu.

Abstract

This term project deals with replacement of webserver in Jenkins CI by server Undertow. The Undertow should be potentially faster than actual state and in general better. In this project there are described general information about components and programs, which are related to this topic. The most important part is analysis of current state and a design of integration. The integration is not accomplished in this term project. It is a subject of follow-up diploma thesis.

Klíčová slova

Jenkins, Undertow, servlet, integrace, kontinuální integrace, Winstone, Jetty, Java

Keywords

Jenkins, Undertow, servlet, integration, continuous integration, Winstone, Jetty, Java

Citace

Jakub Barteček: Integrace serveru Undertow se systémem Jenkins CI, semestrální projekt, Brno, FIT VUT v Brně, 2014

Integrace serveru Undertow se systémem Jenkins CI

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana inženýra Petra Müllera. Další informace mi poskytl pan doktor Vojtěch Juránek, který je zaměstnancem firmy Red Hat®.

.....

Jakub Barteček

14. ledna 2014

Poděkování

Děkuji panu inženýru Petru Müllerovi za vedení mého semestrálního projektu a panu doktoru Vojtěchu Juránkovi za odborné konzultace týkající se zpravovávané problematiky.

© Jakub Barteček, 2014.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod	2
2 Jenkins CI a související nástroje	3
2.1 Jenkins CI	3
2.1.1 Kontinuální integrace a využití Jenkins CI	4
2.1.2 Způsoby použití aplikace	4
2.1.3 Základy práce s Jenkins CI	5
2.1.4 Architektura serveru	6
2.2 Webový server a servlet kontejner	7
2.2.1 Webový server	7
2.2.2 Servlet kontejner	8
2.3 Server Jetty	8
2.4 Servlet kontejner Winstone	9
2.4.1 Nedostatky servlet kontejneru Winstone	9
2.5 Vývoj servlet kontejneru v komunitě Jenkins CI	9
2.6 Server Undertow	10
2.6.1 Vlastnosti serveru	10
2.6.2 Praktický způsob práce se serverem	11
2.6.3 Srovnání serverů Undertow a Jetty	12
3 Analýza současného stavu servlet kontejneru a návrh integrace	14
3.1 Aktuální stav architektury servlet kontejneru v Jenkins CI	14
3.1.1 Architektura Jenkins CI z pohledu servlet kontejneru	14
3.1.2 Průběh komunikace prostřednictvím servlet kontejneru	16
3.2 Zpětná kompatibilita servlet kontejner v Jenkins CI	17
3.2.1 Funkcionality servlet kontejneru	17
3.3 Návrh způsobu integrace serveru Undertow	18
3.3.1 Varianty integrace	18
3.3.2 Zvolení způsobu integrace	19
3.3.3 Zjištěné problémy	19
4 Závěr	21
A Parametry původního servlet kontejneru v Jenkins CI	23

Kapitola 1

Úvod

Tento semestrální projekt se zabývá vylepšením serveru Jenkins CI, který je v praxi velmi využíván pro potřeby průběžného testování softwaru a jeho kontinuální integraci. Vylepšení se týká především webového serveru a *servlet* kontejneru, který je v Jenkins CI integrován. V současném stavu tyto funkce vykonává kombinace nástrojů Winstone a Jetty.

Server Winstone je již neudržovaný a zastaralý nástroj a z tohoto důvodu byl z velké části nahrazen serverem Jetty, který potřebnou funkcionalitu poskytuje. Server Jetty je poměrně komplexní projekt a nabízí mnoho funkcionality, ale na druhou stranu jeho rozsah nedovoluje poskytovat maximální rychlost.

V současné době vznikl nový webový server Undertow, který si klade za cíl být co nejjednodušší a nejrychlejší a mohl by být přínosný a vhodný pro Jenkins CI. Jelikož tento server je nový a je sponzorován firmou Red Hat, tak lze předpokládat, že jeho vývoj bude nadále pokračovat a nebude zastarávat.

Cílem této práce je nahradit server Jetty a případně i server Winstone pomocí serveru Undertow a integrovat jej se serverem Jenkins CI. Při integraci je kladen důraz na snahu zachovat zpětnou kompatibilitu s původním řešením.

V rámci tohoto semestrálního projektu jsou nejprve rozebrány potřebné informace týkající se jednotlivých nástrojů a plánované integrace. Následně je detailně analyzována architektura Jenkins CI a způsob jeho integrace se servery Winstone a Jetty. V navazující části je diskutována varianta nahrazení pouze serveru Jetty a varianta nahrazení serveru Jetty i Winstone pomocí serveru Undertow. Z provedené analýzy je zvolena jedna varianta, která je vybrána pro následnou integraci.

Samotná implementace integrace není předmětem tohoto semestrálního projektu a bude provedena až v navazujícím diplomovém projektu. V rámci diplomového projektu bude také provedeno testování výkonu modifikovaného systému Jenkins CI a celkové shodnocení navržených a provedených změn. Zadání této práce bylo vytvořeno firmou Red Hat s jejíž spoluprací tato diplomová práce vzniká. Pokud budou výsledky práce dobré, tak její výstup může být následně využit i v praxi.

Kapitola 2

Jenkins CI a související nástroje

Tato kapitola se zaměřuje na teoretické základy práce, které je nutné nebo vhodné znát pro pochopení zpracovávané problematiky. Je zde detailněji popsán systém Jenkins CI (kapitola 2.1) ke kterému se tato práce přímo váže. S ním je spojeno seznámení se servery Winstone (kapitola 2.4) a Jetty (kapitola 2.3), jejichž kombinace je současně v Jenkins CI integrována. Po popisu těchto serverů je rozebírán průběh jejich využití v servlet kontejneru Jenkins CI a průběh jeho vývoje (kapitola 2.5). V kapitole 2.2 jsou vysvětleny často zde používané pojmy *servlet kontejner* a webový server.

Větší důraz je dále věnován serveru Undertow (kapitola 2.6), který byl vybrán jako nový webový server pro Jenkins CI. V tomto případě je provedena hlubší studie tohoto nástroje, aby na jejím základě bylo možné pochopit a provést samotnou integraci s Jenkins CI, která je jádrem této práce.

Uvedené informace mají spíše informativní charakter, aby poskytly ucelený úvod do zkoumané problematiky. Jsou zaměřeny především na informace týkající se samotné integrace. Pro případné získání detailnějších informací jsou uvedeny patřičné zdroje, kde je lze nalézt.

2.1 Jenkins CI

Jenkins CI je komunitní open source nástroj pro kontinuální integraci softwaru, který je vyvíjen pod svobodnou licenci MIT¹ [2]. Je velmi populární a využíván malými i velkými firmami jako je například firma Red Hat, kde tento program běží na stovkách serverů. Původní název tohoto projektu je Hudson². Když se jeho vývoje ujala firma Oracle, tak se projekt rozštěpil a vznikla jeho komunitní verze, kterou je projekt Jenkins CI. Přesto se v některých částech tohoto projektu stále objevuje název Hudson, ale jedná se pouze pozůstatek z původního projektu.

Zkratka CI je z anglického spojení *continuous integration*, což lze do češtiny přeložit jako kontinuální nebo průběžná integrace. Krátké seznámení s touto metodologií je v následující kapitole.

Informace v této kapitole byly čerpány především z knihy [7], kde lze nalézt další informace o serveru Jenkins CI, a také z webové stránky projektu [10].

¹Licence MIT: <http://opensource.org/licenses/MIT>

²Webové stránky projektu Hudson: <http://hudson-ci.org/>

2.1.1 Kontinuální integrace a využití Jenkins CI

V minulosti byla integrace programu do výsledného produktu velmi náročným procesem a často ztraceným časem. S vydáním každé verze programu se musel postup probíhající před vydáním produktu opakovat a pro vývojářský tým to prakticky znamenalo zdržení. Pokud se v tomto procesu odhalil nějaký problém (což bylo běžné), tak jeho řešení bylo z důvodu nedostatku času a jeho pozdního objevení mnohem problematičtější než kdyby byl tento problém odhalen dříve.

Kontinuální integrace je moderní přístup k vývoji softwaru, který mění způsob přemýšlení nad celým procesem vývoje a snaží se předcházet problémům popsáným výše a především ušetřit čas. V tomto přístupu je využíván nějaký kvalitní nástroj, který automatizovaně provádí specifikované kroky, které provázejí integraci softwaru a jeho vydání.

Jedním z nástrojů poskytujících podporu pro kontinuální integraci při vývoji softwaru je server Jenkins CI.

Základními možnostmi, které umožňuje Jenkins CI nakonfigurovat, jsou:

- Spouštění integračních a jednotkových testů v přesně definovaném čase (např. v noci, kdy jsou servery méně vytížené)
- Spuštění integračních a jednotkových testů při změně ve verzovacím systému. Jenkins CI dokáže zaznamenat změnu v repozitáři, stáhnout si změny a spustit testování
- Shromažďování a vyhodnocování metrik vývoje softwaru
- Spuštění akceptačních testů
- Informování e-mailem o testech, které skončily chybou
- Automatické nahrání nové verze produktu na server

Uživatelé mohou kdykoliv přidat využití libovolné funkcionality systému a neopakovat stále stejné kroky.

2.1.2 Způsoby použití aplikace

Celý program je napsán v jazyce Java a je tedy plně přenositelný mezi platformami. Architektura je navržena tak, aby byla lehce rozšiřitelná pomocí tzv. *pluginů*, kterých je pro něj vytvořené velké množství.

Jenkins CI je určen pro běh na serveru a je dostupný přes webové rozhraní (ale může být samozřejmě spuštěn na libovolném osobním počítači). Komunikuje pomocí protokolu *HTTP*, který je založen na modelu *požadavek-odpověď* (angl. *request-response*). Jelikož tento způsob komunikace je velmi běžný, tak není v programu přímo implementován, ale využívá k němu externí nástroje. Dalším nástrojem, který pro svou činnost Jenkins potřebuje, je servlet kontejner ve kterém samotná aplikace poběží. Tento pojem je blíže objasněn v kapitole 2.2.

Existují dvě možnosti jak spustit server Jenkins CI:

1. Může běžet na libovolném *Java EE* aplikačním serveru [1] jako jsou například servery JBoss³ anebo Glasfish⁴. Jenkins CI se standardně nasadí na server (dle zvyklostí

³Více informací o serveru viz <http://www.jboss.org/jbossas/>

⁴Více informací o serveru viz <http://www.oracle.com/technetwork/middleware/glassfish/>

konkrétního serveru) a poté je s ním možné pracovat. V tomto případě veškerou nízkourovňovou komunikaci pomocí *HTTP* i práci servlet kontejneru zajišťuje aplikační server.

2. Pokud nechceme nebo nemůžeme spouštět Jenkins CI na aplikačním serveru, tak jej lze spustit přímo z vytvořené `.war` archivu (překladem se zabývá kapitola 2.1.3). V tomto případě se o práci servlet kontejneru i webového serveru komunikujícího protokolem *HTTP* stará kombinace nástrojů Winstone a Jetty, které jsou přímo integrovány do serveru Jenkins CI.

Nahrazení těchto dvou nástrojů (nebo pouze serveru Jetty) a zajištění vykonávání této činnosti je hlavním cílem této práce. Záměrem je tedy nahradit server Jetty a případně i server Winstone pomocí zvoleného nového serveru Undertow. Studie těchto nástrojů a jejich rozbor je předmětem následujících kapitol.

Pro tuto práci je důležitý druhý způsob používání Jenkins CI a proto další informace se budou přímo vázat k němu.

2.1.3 Základy práce s Jenkins CI

Pro přeložení a spuštění aplikace je potřeba pracovat z příkazové řádky nebo provést instalaci nějakým dávkovým souborem (skriptem). Aplikaci je možné stáhnout připravenou přímo ze stránek projektu [10], ale pro tuto práci je potřeba pracovat s aplikací ze zdrojových souborů. Aktuální verze aplikace je dostupná na serveru GitHub⁵.

Po stažení zdrojových souborů je potřeba provést překlad aplikace. Pro tento automatizovaný překlad se používá nástroj Maven⁶.

Překlad aplikace bez spuštění jednotkových testů lze provést tímto příkazem:

```
mvn clean install -pl war -am -DskipTests
```

Pokud bychom chtěli spustit jednotkové testy aplikace, tak jednoduše vynecháme poslední přepínač. Nejjednodušším způsobem kontroly provedených změn v programu je právě spuštění automatizovaných testů a proto je tato možnost velmi důležitá pro plánovanou integraci se serverem Undertow. Kromě jednotkových testů obsahuje Jenkins CI také integrační testy, které jdou více do hloubky aplikační logiky a jsou schopny odhalit větší množství chyb, ale na druhou stranu jejich vykonání trvá dlouhou dobu (řádově více než hodinu). Integrační testy lze spustit příkazem:

```
mvn clean package
```

Další nevýhodou integračních testů je, že je běžný stav, kdy některé testy aplikace, která je překládána z aktuálních zdrojových souborů, končí s chybou. Proto je při kontrole nutné porovnávat výsledky testů před provedením změn a po jejich provedení. Tento způsob kontroly aplikace bude využit při hodnocení navazující diplomové práce.

Po úspěšném překladu aplikace vznikne v adresáři `./war/target/` archiv `jenkins.war`, který obsahuje celou přeloženou webovou aplikaci včetně nástrojů na kterých závisí. Z tohoto archivu je možné aplikaci přímo spustit pomocí příkazu:

⁵Adresa aktuální verze Jenkins CI: www.github.com/jenkinsci

⁶Více informací o nástroji Maven lze získat na stránce <http://maven.apache.org/>

```
java -jar jenkins.war
```

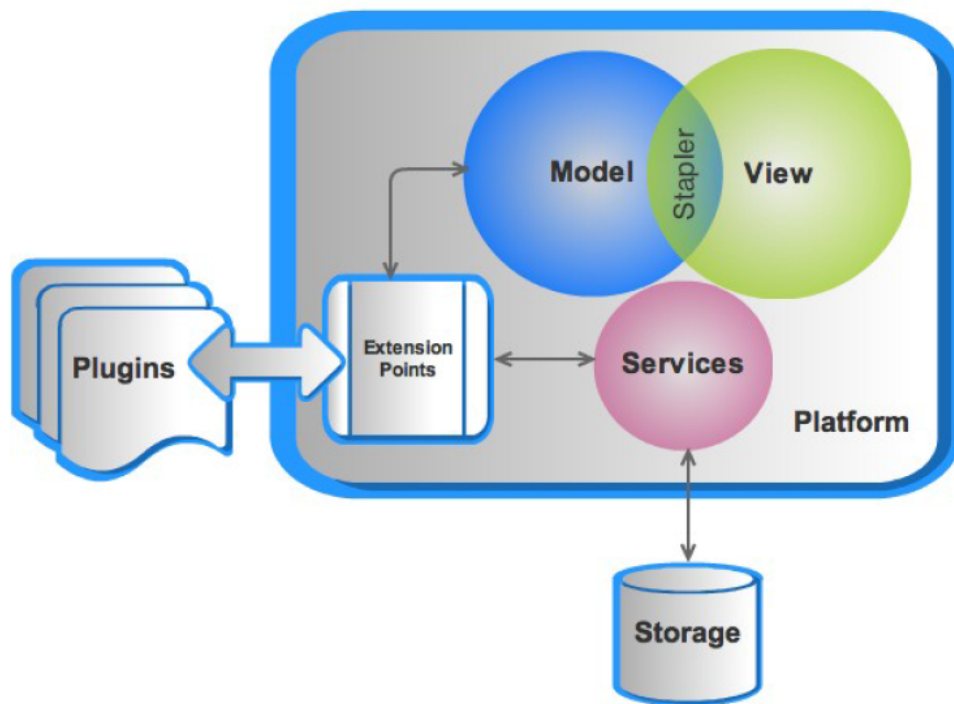
Je možné nastavit ještě další parametry při spuštění programu, které lze vypsat pomocí přidání parametru `--help`.

Pro práci se spuštěnou instancí aplikace se používá především webové rozhraní. Standardně aplikace komunikuje pomocí protokolu *TCP* na portu 8080. Je tedy možné se na lokálním počítači připojit do aplikace zadáním URL adresy `http://localhost:8080` do webového prohlížeče.

Pokud se aplikaci povede spustit, tak je již možné libovolně pracovat pouze s prohlížeče. Detaily práce s aplikací Jenkins CI lze nalézt v této knize [7], ale tyto informace jsou již nad rámec této publikace.

2.1.4 Architektura serveru

Architektura systému Jenkins CI je poměrně komplikovaná a pro tuto práci není nutné ji detailně celou znát. Bude popsána na vysoké úrovni abstrakce a zaměří se pouze na komponenty, které se přímo týkají této práce a budou dále v textu odkazovány nebo blíže rozebírány. Přehled architektury je na obrázku 2.1.



Obrázek 2.1: Přehled architektury serveru Jenkins CI [6]

Základem architektury je část *Model*, což jsou objekty, které obsahují stav a data aplikace. Každý model je přímo navázán na konkrétní URL adresu s tím, že kořenový model (dostupný pod URL „/“) je pevně daná instance s názvem *Hudson*. Data z objektů modelu jsou následně zobrazovány ve webovém rozhraní aplikace. Pro toto zobrazování je využita technologie Jelly.

Velmi podstatnou součástí aplikace je prvek *Stapler*. Tento objekt provádí konkrétní propojování požadavků dle zadané URL s patřičnými objekty z modelu a spouštění vykonávání jejich metod. Stapler je jediným servletem, který je v aplikaci Jenkins CI zaveden do servlet kontejneru při spuštění aplikace (pojmy servlet a servlet kontejner jsou vysvětleny v kapitole 2.2.2). Povědomí o jeho činnosti je potřebné, protože s ním bude v této práci dále pracováno. Průběh vybírání patřičných metod pomocí této komponenty je přesně definován a lze jej najít v uživatelské příručce⁷, ale není nutné jej zde rozebírat.

Architektura serveru je přizpůsobena tak, aby byla snadno rozšiřitelná pomocí rozšiřujících modulů (angl. *plugins*). Popis technologie Jelly i tvorba rozšiřujících modulů je nad rámec této práce a lze tyto informace najít v odkazované literatuře.

Informace v této kapitole byly čerpány z tohoto dokumentu [6] a z webových stránek projektu Stapler [9].

2.2 Webový server a servlet kontejner

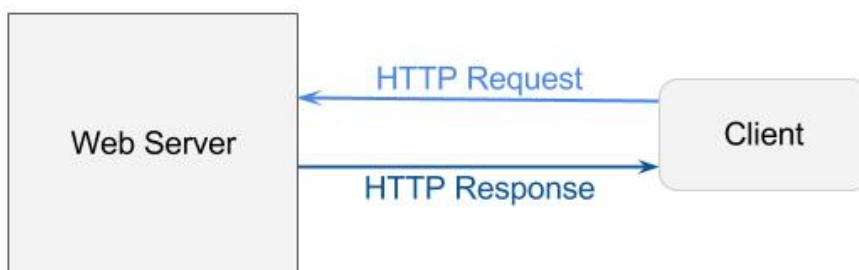
V této kapitole jsou vysvětleny pojmy webový server a servlet kontejner, které se na mnoha místech této práce objevují. Porozumění těmto pojmům je důležité, protože bez jejich znalosti by následující kapitoly byly obtížněji pochopitelné.

Informace zde uvedené byly čerpány z článku [8].

2.2.1 Webový server

Webový server je program, který zprostředkovává komunikaci přes síť s klienty, kteří se k němu připojí a požadují po něm nějaká data. Tato komunikace probíhá pomocí protokolu *HTTP*, který je založen na modelu *požadavek-odpověď* (angl. *request-response*). Model této komunikace je zachycen na obrázku 2.2.

Typický způsob komunikace webového serveru je, že klient pomocí URL adresy specifikuje požadavek na nějaká data a server mu v odpovědi tato data pošle. Pokud by neexistoval za webovým serverem nějaký další program, tak by webový server vždy odpovídal na stejný požadavek stále stejnou odpovědí.



Obrázek 2.2: Model komunikace webového serveru s klientem [8]

⁷ Způsob zpracovávání požadavků komponentou Stapler: <http://stapler.kohsuke.org/reference.html>

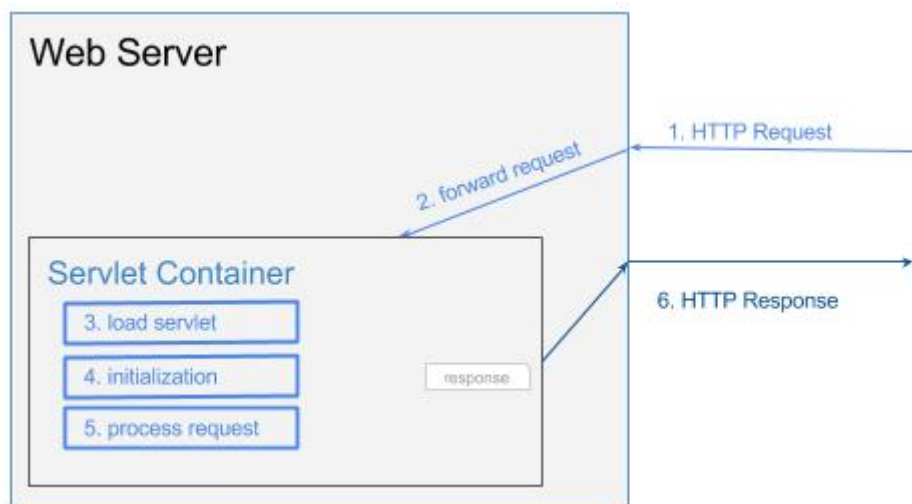
2.2.2 Servlet kontejner

Jelikož dostávat stále stejná statická data při stejném požadavku není dostatečná funkcionality serveru, tak existují způsoby jak zajistit dynamickou práci s daty na serveru a tudíž i poskytovat měnící se odpovědi na stejný dotaz. Jedním ze způsobů jak této funkce docílit je využití tzv. *servletů*, které jsou navrženy pro programovací jazyk Java.

Servlet je standardní program (konkrétně jedna třída) napsaný v jazyce Java, který implementuje rozhraní `javax.servlet`. Implementace tohoto rozhraní ho zavazuje k definování několika metod, ale jinak se jedná o běžnou třídu z které jsou při zpracovávání požadavků vytvářeny její instance. Po obdržení nějakého požadavku provádí zpracování vstupních dat a následně odeslání patřičné odpovědi.

Základní myšlenkou servlet kontejneru je umožnit dynamicky vytvářet odpovědi (často webové stránky) pomocí vykonávání servletů. Samotný servlet kontejner je program, který poskytuje běhové prostředí pro vykonávání servletů, zajišťuje jejich vytváření, vykonávání a odstraňování. Dále se také podílí na zpracovávání HTTP požadavků, které jsou mu předávány od webového serveru.

Popsaný způsob fungování servlet kontejneru je znázorněn na obrázku 2.3.



Obrázek 2.3: Ukázka způsobu činnosti servlet kontejneru a jeho spolupráce s webovým serverem [8]

2.3 Server Jetty

Jednou z komponent, které jsou aktuálně integrovány do serveru Jenkins CI, je webový server Jetty. Tento server je open source projektem vyvíjeným pod licencemi Eclipse⁸ a Apache⁹. Je využíván velkým množstvím nástrojů jako jsou například *Eclipse IDE*¹⁰ nebo *Google AppEngine*¹¹.

⁸Licence Eclipse je dostupná na adrese <http://www.eclipse.org/legal/epl-v10.html>

⁹Licence Apache je dostupná na adrese <http://www.apache.org/licenses/LICENSE-2.0.html>

¹⁰Webové stránky nástroje Eclipse IDE <http://www.eclipse.org/>

¹¹Webové stránky platformy Google AppEngine <https://developers.google.com/appengine/>

Jetty je webový server a poskytuje funkcionalitu servlet kontejneru dle specifikace verze 3.0. Je jej možné využít také jako webového klienta pro komunikaci se servery. Komunikace klientů i serverů využívajících Jetty probíhá asynchronně. Návrh serveru umožňuje samostatné spouštění aplikací i jeho integraci do jiné aplikace.

Kromě těchto základních možností obsahuje řadu souvisejících technologií jako jsou SPDY, webové sokety¹² (angl. *websocket*), JNDI, OSGi, JMX a další. Informace v této kapitole byly čerpány z webových stránek serveru Jetty [12], kde lze nalézt další informace o zmíněných technologiích a o tomto serveru.

Aktuální využití serveru Jetty v aplikaci Jenkins CI je podrobněji rozebráno v kapitole 2.5 a jeho srovnáním se serverem Undertow se zabývá kapitola 2.6.3.

2.4 Servlet kontejner Winstone

Další komponentou serveru Jenkins CI je servlet kontejner Winstone. Winstone je velmi jednoduchý a poskytuje funkcionalitu servlet kontejneru aniž by byl zatížen velkým množstvím požadavků, které jsou ve specifikaci jazyka Java EE. Nikdy neposkytoval veškeré služby servlet kontejneru, které specifikace jazyka definuje. V jeho názvu je zahrnutý pouze pojem servlet kontejner, ale tato aplikace vykonává i služby webového serveru odpovídající popisu v kapitole 2.2.

Hlavními cíli projektu bylo poskytovat funkcionalitu servlet kontejneru pouze pro jednu aplikaci, což je opačný přístup než u běžných aplikačních serverů jako jsou Glasfish, JBoss, aj. Díky jeho omezeným službám je jeho velikost velmi malá a umožňuje jednoduchou integraci s cílovou aplikací.

Uvedené informace byly čerpány z oficiální stránky projektu Winstone [4].

2.4.1 Nedostatky servlet kontejneru Winstone

Původní myšlenka jednoduchého servlet kontejneru byla pro projekt Jenkins CI zajímavá, ale kontejner Winstone má několik zásadních nedostatků kvůli kterým bylo časem jeho využití v Jenkins CI problematické [3].

Vývoj projektu Winstone již před delší dobou ustal a tím pádem nebyla poskytována žádná další podpora pro řešení a opravování objevených nedostatků a chyb. Značné množství bezpečnostních chyb objevených v projektu Jenkins CI bylo právě způsobeno tímto servlet kontejnerem.

Zpočátku možnosti servlet kontejneru postačovaly, ale časem je potřeba, aby byly přidávány nové funkcionality, které odpovídají aktuálním trendům a novým specifikacím. Příkladem mohou být nové specifikace servlet kontejneru nebo vývoj nových technologií jako webové sokety.

2.5 Vývoj servlet kontejneru v komunitě Jenkins CI

Po ukončení vývoje projektu Winstone se musela o jeho potřebné úpravy starat komunita Jenkins CI. Takováto práce je pro komunitu velmi zatěžující a naprosto neefektivní. Byly prováděny především nutné opravy bezpečnostních chyb v kontejneru, ale jinak aplikace dále degenerovala. Pro tento vývoj vznikl v projektu Jenkins CI nový repozitář, který vycházel

¹²Oficiální stránky specifikace webových soketů: <http://www.websocket.org/>

z původní verze servlet kontejneru Winstone¹³. Z tohoto zdroje a z článku o integraci Jetty s kontejnerem Winstone [3] jsou čerpány uváděné informace.

Vývoj původního servlet kontejneru Winstone probíhal uvedeným způsobem až do verze *0.9.10-jenkins-47*. Následně byl tento způsob vývoje zastaven a do kontejneru Winstone byl integrován webový server a servlet kontejner Jetty. Tímto krokem vznikla interní verze servlet kontejneru *Winstone 2.0*. Velká část kódu kontejneru Winstone byla odstraněna a veškerá činnost webového serveru a servlet kontejneru je nyní vykonávána pomocí serveru Jetty. Z původního kontejneru Winstone zůstal způsob zpracovávání a nastavování parametrů. Tato změna proběhla poměrně narychlo a nebyla detailně otestována. Obsahuje zřejmě ještě množství nepotřebného kódu a samotná implementace není moc předhledná.

Těsně před integrací serveru Jetty s Jenkins CI vznikalo zadání tohoto diplomového projektu, které mělo za cíl výrazně zlepšit aktuální stav servlet kontejneru v projektu Jenkins CI. Během formulace zadání byl servlet kontejner v projektu přepracován a proto muselo být zadání upraveno. I po této změně byly stále důvody pro nahrazení stávajícího servlet kontejneru serverem Undertow. Aktuální situace již není tak kritická jako v předchozí verzi, ale může přinést ještě další zlepšení.

2.6 Server Undertow

Undertow je nový webový server, který vzniká za podpory firmy Red Hat a její sekce JBoss, a je vytvořen v jazyce Java. Vzniká především aby byl výchozím webovým serverem v aplikačním serveru WildFly 8. V současné době je ve fázi beta verze programu, ale je již plně použitelný.

Informace v této kapitole byly čerpány z webových stránek projektu [11]. Jelikož je aplikace poměrně nová, tak k ní existuje jen poměrně nedostatečná dokumentace. Některé informace z této kapitoly musely být čerpány z vygenerované projektové dokumentace.

2.6.1 Vlastnosti serveru

Server Undertow je zaměřen na to, aby byl plně integrovatelný do libovolných aplikací a aby byl co nejmenší a nejjednodušší. Samotný archiv s jádrem aplikace je menší než 1MB a při běhu aplikace potřebuje méně než 4MB dynamicky alokované paměti.

Je navržen takovým způsobem, aby při implementaci mohl uživatel využít jen část aplikace, kterou nutně potřebuje, a patřičně si ji upravit pro své vlastní potřeby. Tohoto přístupu je dosaženo kombinováním a řetězením obslužných funkcí (angl. *handler*), které server poskytuje. Díky tomuto přístupu je server velmi flexibilní a v jeho důsledku také patřičně rychlý, protože uživatele nebrzdí funkcionality serveru, které nutně nepotřebuje a nevyužívá.

Při komunikaci umožňuje server podporu jak pro asynchronní, tak pro synchronní komunikaci. Dalšími funkcionalitami, které server poskytuje, jsou možnost integrace servlet kontejneru odpovídajícího specifikaci verze 3.1, využití plné podpory webových soketů nebo podpory technologie *HTTP upgrade*.

¹³Repozitář, kde komunita Jenkins CI provádí úpravy projektu Winstone: <https://github.com/jenkinsci/winstone>

2.6.2 Praktický způsob práce se serverem

Server Undertow obsahuje velké množství funkcionalit, které lze využít při integraci do cílové aplikace. V následujícím textu bude na dvou příkladech ukázán způsob implementace jednoduchého serveru, který umožní pochopit jakým způsobem se se serverem Undertow pracuje. Obsah těchto příkladů bude jistě využit také při integraci s Jenkins CI.

Prvním příkladem je ukázková aplikace, která pouze naslouchá na zvoleném portu a na jakoukoliv příchozí odpověď odpoví typickým textem „Hello World“. Celý kód této aplikace vypadá následovně:

```
public class HelloWorldServer {

    public static void main(final String[] args) {
        Undertow server = Undertow.builder()
            .addListener(8080, "localhost")
            .setHandler(new HttpHandler() {
                @Override
                public void handleRequest(final HttpServerExchange exchange)
                    throws Exception {
                    exchange.getResponseHeaders()
                        .put(Headers.CONTENT_TYPE, "text/plain");
                    exchange.getResponseSender().send("Hello World");
                }
            }).build();
        server.start();
    }
}
```

Třída `Undertow` vždy reprezentuje instanci samotného webového serveru a je ji nutné definovat prakticky vždy. Funkcí `addListener` je serveru sděleno na kterém portu a síťové adrese má naslouchat příchozím požadavkům. Pomocí funkce `setHandler` je k danému portu přiřazena metoda, která se stará o zpracovávání požadavků zaslaných na server a odesílání odpovědí. V tomto případě pouze jen odpoví již zmíněným řetězcem „Hello World“. Samotné vytvoření požadavku a jeho odeslání provede tato metoda pomocí entity `HttpServerExchange`, která zpracovává komunikaci a obsahuje o ni množství další informací (např. informace z hlavičky dotazu).

Je důležité poznamenat, že pro každé naslouchání na nějakém portu musí být vytvořena nová instance serveru.

Druhý příklad ukazuje jakým způsobem aplikace umožňuje přidat funkcionalitu servlet kontejneru do webového serveru. V následujícím příkladu je zobrazen pouze efektivní kód aplikace.

```
DeploymentInfo servletBuilder = deployment()
    .setClassLoader(ServletServer.class.getClassLoader())
    .setContextPath("/app")
    .setDeploymentName("app.war")
    .addServlets(
```



```

        servlet("MyServlet", CustomServlet.class)
            .addInitParam("paramKey", "paramValue")
            .addMapping("/servlet"));

DeploymentManager manager = defaultContainer().addDeployment(servletBuilder);
manager.deploy();

Undertow server = Undertow.builder()
    .addListener(8080, "localhost")
    .setHandler(manager.start())
    .build();
server.start();

```

V prvním kroku ukázkového příkladu se provádí nastavování webové aplikace, která bude zavedena v servlet kontejneru. Tyto informace se ukládají do entity `DeploymentInfo`. Nejprve je nastavena aplikaci hlavní třída kontejneru pomocí metody `setClassLoader`. Metodou `setContextPath` je definován tvar URL adresy pod kterou bude na serveru tato aplikace dostupná. Důležitá je metoda `addServlets` pomocí které se přidají jednotlivé součásti aplikace a nastaví se jejich dostupnost v aplikaci metodou `addMapping`.

Po nastavení konfigurace aplikace proběhne metodami `addDeployment` a `deploy` připravení aplikace po spuštění. Ve zbytku kódu je spuštěna instance serveru obdobným způsobem jako byla ukázána v předchozím příkladu.

2.6.3 Srovnání serverů Undertow a Jetty

Stávající servlet kontejner v Jenkins CI je sice tvořen kombinací kontejneru Winstone a serveru Jetty, ale veškerou časově kriticky náročnou práci při běhu aplikace vykonává server Jetty. Proto pokud chceme srovnávat aktuální a plánované řešení servlet kontejneru v Jenkins CI, tak bychom měli srovnávat server Jetty se serverem Undertow.

Budeme tedy srovnávat servery Undertow a Jetty, jejich výhody a nevýhody vzhledem k využití v systému Jenkins CI:

- **Rychlost:** Server Undertow je aktuálně ještě stále ve verzi *beta*, ale už přesto existují testy v kterých se ukázal být výkonnější než server Jetty. V tomto testování¹⁴ byl server Undertow i 3,5 krát rychlejší než server Jetty v počtu zpracovaných požadavků za jednotku času. Při porovnávání doby odezvy serveru na požadavek dosáhl server Undertow až třetinového času než server Jetty.

Po dokončení diplomového projektu a provedení plánované integrace bude uskutečněno vlastní porovnání těchto dvou implementací s cílem potvrdit nebo vyvrátit tyto výsledky.

- **Spolehlivost:** Dalším důležitým aspektem je spolehlivost daného serveru. Server Jetty má za sebou již dlouhou historii a je integrován ve velkém množství různých aplikací¹⁵, což mu dodává velkou důvěryhodnost a lze očekávat, že bude pracovat velmi spolehlivě.

¹⁴Porovnání rychlosti, odezvy a celkové výkonnosti serverů Undertow, Jetty a jiných: <http://www.techempower.com/benchmarks/#section=data-r8&hw=ec2&test=plaintext>

¹⁵Aplikace, které využívají server Jetty: <http://www.eclipse.org/jetty/powered/>

U serveru Undertow ještě nebyla dokončena první finální verze, takže lze předpokládat, že může obsahovat ještě drobné nedostatky, které budou časem opravovány. Jelikož tento server bude integrován v novém aplikačním serveru WildFly 8, tak lze předpokládat, že postupem času bude také velmi spolehlivý. Nicméně v tomto aspektu je server Jetty zřejmě aktuálně lepší.

- **Rychlost spuštění:** Velmi často je u serverů hodnocena hlavně jejich rychlost a výkonnost v zátěži. Je běžné, že nebývá kladen důraz na velmi rychlé spuštění serveru. Pro standardní běh serveru, kdy bývá spuštěn jednou za několik týdnů či měsíců po nějakých úpravách aplikace, není doba spuštění zásadní. Pokud se na to podíváme z pohledu testování aplikace, kdy jsou spouštěny integrační testy a pro každý test musí být znovu spuštěn server, tak je rychlost nastartování serveru velmi podstatná.

Cílem serveru Undertow je být minimalistickým a velmi rychlým řešením. Server Jetty poskytuje kvalitní podporu pro běh aplikací, ale je rozsáhlejší a není jeho prioritou, aby byl extrémně rychle spuštěn. Lze usuzovat, že rychlost spuštění bude u serveru Undertow výrazně nižší než u serveru Jetty. Tato skutečnost bude testována při vyhodnocování výsledků této diplomové práce.

V aktuálním stavu Jenkins CI běží integrační testy více než hodinu. Pokud by byl servlet kontejner spuštěn za poloviční dobu, tak by mohla být doba testování zkrácená také téměř na polovinu.

- **Velikost:** Jelikož je servlet kontejner přímo integrován do archivu ve kterém je systém Jenkins CI distribuován, tak komunita dbá na to, aby přidávané komponenty nebyly příliš velké a průběžně sleduje aktuální velikost výsledného archivu¹⁶.

Po integraci Jetty vzrostla velikost servlet kontejneru o 1,5 MB na celkovou velikost 1,8 MB. U serveru Undertow je uváděno, že jeho archiv má méně než 1 MB, ale konečná velikost po integraci bude také záležet na využitých komponentách a rozsahu implementace. Přesto z pohledu velikosti výsledného archivu se jeví server Undertow jako výhodnější, ale rozdíl oproti serveru Jetty není zásadní. Rozdíl je zřejmě přibližně 1MB, což jsou necelé 2% z aktuální velikosti Jenkins CI (cca 66 MB).

- **Konfigurovatelnost:** Server Undertow je velmi flexibilní a způsob jeho návrhu, který byl popsán výše, umožňuje provádět mnoho různých úprav své činnosti dle potřeby a využít pouze ty části, které potřebujeme. Tato filozofie je velmi blízká filozofii projektu Jenkins CI.

Server Jetty je oproti tomu robustnější a rozsáhlejší, ale neumožňuje tak velké přizpůsobování potřebám uživatele jako například využití jen několika malých částí jeho funkčnosti či jejich kombinování.

¹⁶Stránka, kde je v systému Jenkins CI automatizovaně sledována velikost archivu aplikace:
<https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+WAR+Size+Tracker>

Kapitola 3

Analýza současného stavu servlet kontejneru a návrh integrace

Tato kapitola se zabývá důkladnější analýzou a zkoumáním architektury aplikace Jenkins CI z pohledu jejího vestavěného kontejneru a jeho možných úprav (kapitola 3.1). V další části jsou konkretizovány jednotlivé činnosti, které současný servlet kontejner provádí a které musí nová implementace také poskytovat (kapitola 3.2). Poslední část této kapitoly diskutuje možné varianty integrace serveru Undertow do Jenkins CI, jejich výhody a nevýhody (kapitola 3.3). Jako výstup této analýzy je zvolen způsob integrace, který bude následně implementován.

Jelikož k této problematice je minimum oficiálních zdrojů, které by danou problematiku blíže popisovaly, tak podstatná část zde uváděných informací byla čerpána přímo ze zdrojových kódů aplikace a komponent Jenkins CI.

3.1 Aktuální stav architektury servlet kontejneru v Jenkins CI

V následujících dvou podkapitolách bude blíže analyzována architektura Jenkins CI z pohledu servlet kontejneru. Tato analýza je velmi důležitá pro pochopení návazností jednotlivých komponent, které budou později upravovány.

Jak již bylo uvedeno v kapitole 2.5, současný servlet kontejner v Jenkins CI se skládá z nástrojů Winstone a Jetty, které byly také popisovány v předchozích kapitolách. Velká část servlet kontejneru Winstone byla odstraněna a zůstala pouze část, která provádí zpracování parametrů programu po spuštění aplikace. Činnost webového serveru, servlet kontejneru a další vykonává server Jetty.

3.1.1 Architektura Jenkins CI z pohledu servlet kontejneru

Na vysoké úrovni pohledu lze architekturu serveru Jenkins CI rozdělit do tří částí:

- **Jádro aplikace** do kterého patří nejnutnější základní komponenty systému a části, které jsou pro Jenkins CI specifické a vznikají v rámci tohoto projektu.
- **Přídavné moduly** aplikace, které jsou do ní dynamicky přidávány jako archivy javo- vých programů `.jar`. Tyto součásti jsou většinou nutné pro standardní běh aplikace a je s nimi aplikace běžně dodávána (teoreticky lze aplikaci spustit např. bez servlet kontejneru pomocí aplikačního serveru, ale toto je spíše ojedinělý případ).

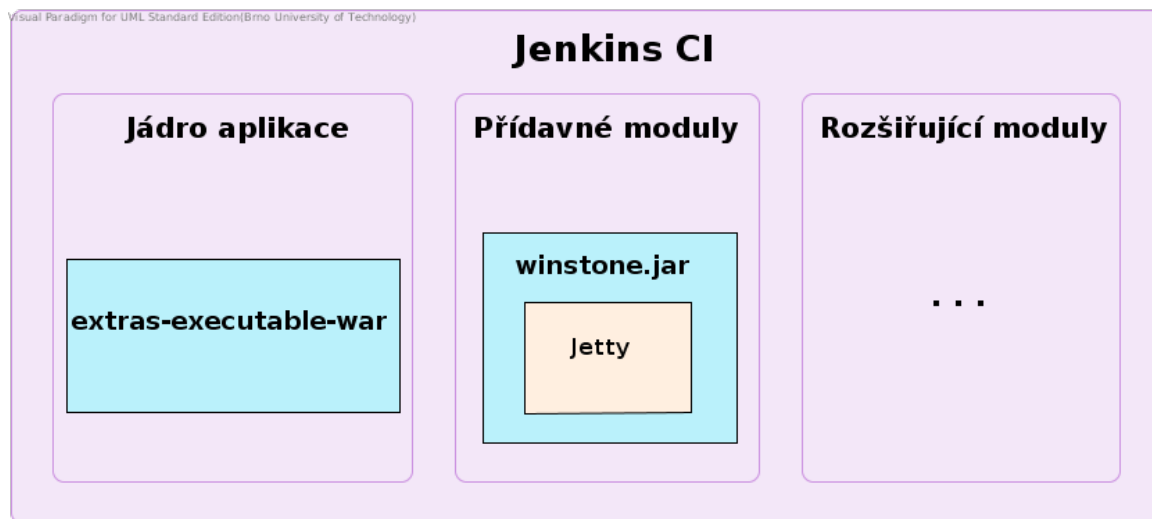
Komponenty z této kategorie pocházejí typicky z externích projektů. Do této kategorie také spadá servlet kontejner, který je aktuálně do aplikace přidáván pod názvem `winstone.jar`.

- **Rozšiřující moduly** (angl. plugins) jsou samostatné menší programy, které nějakým způsobem přidávají funkcionalitu serveru Jenkins CI. Typicky nejsou dodávány s aplikací a uživatel si je může stáhnout nebo si nějaký vlastní modul vytvořit.

Samotná aplikace se dodává jen s těmi nejn nutnějšími součástmi a ponechává na uživatelích, které další funkcionalitu si do systému doinstalují. V současné době již existují stovky takových modulů, které jsou volně ke stažení.

V následujícím textu bude při popisu interních součástí zdrojových kódů použita notace ve tvaru `Název_třídy::Název_metody`.

Architektura Jenkins CI z pohledu servlet kontejneru je znázorněna na obrázku 3.1. Jsou zde zachyceny především komponenty, které přímo souvisejí s jeho činností.



Obrázek 3.1: Přehled architektury systému Jenkins CI z pohledu vestavěného servlet kontejneru

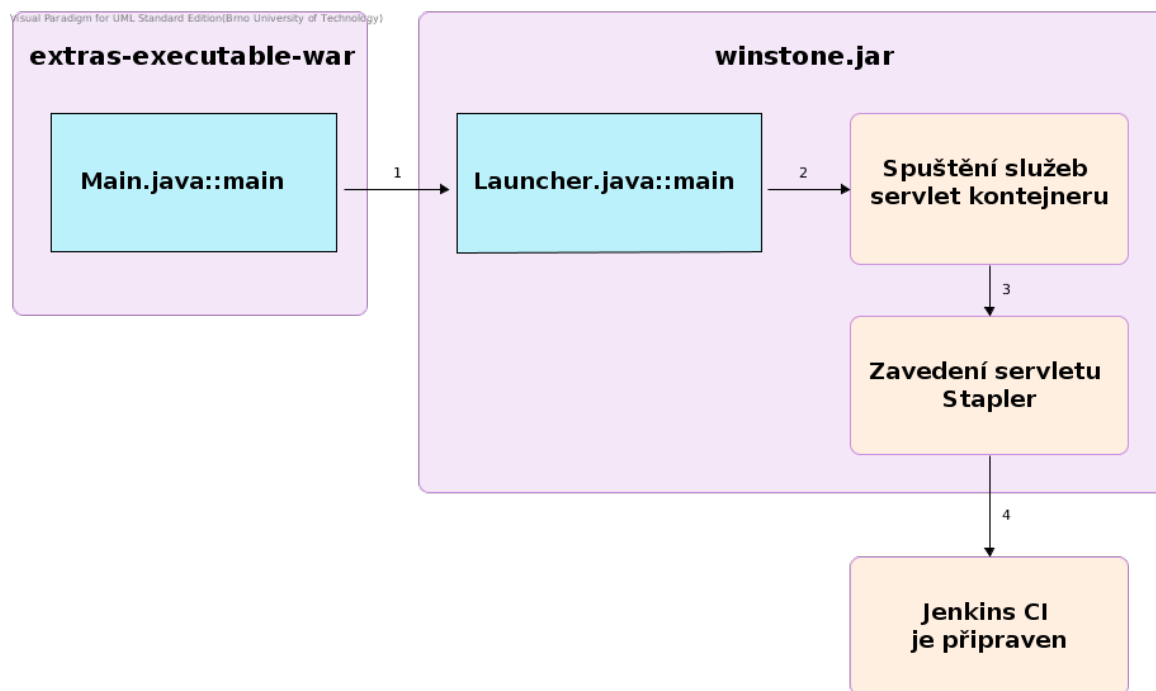
Archiv `winstone.jar` obsahuje servlet kontejner pro Jenkins CI. Název je nyní sice mírně matoucí (způsoben předchozí implementací), ale aktuálně jsou součástí tohoto archivu komponenty Winstone a Jetty.

Velmi podstatnou součástí aplikace z pohledu servlet kontejneru je komponenta `extras-executable-war`. Tato součást aplikace je sice malá, ale obsahuje metodu `main`, kterou se aplikace spouští (pokud není spuštěna v aplikačním serveru). Hlavním úkolem této komponenty je právě spustit servlet kontejner a tím spustit i celou aplikaci. Tento proces spuštění aplikace je zobrazen na obrázku 3.2.

Nejprve je spuštěna vstupní metoda celé aplikace `Main.java::main` z komponenty `extras-executable-war`. Po počáteční inicializaci je předáno řízení aplikace metodě `Launcher.java::main` z archivu `winstone.jar`, která provede nastartování a zavedení servlet kontejneru. Následně je provedena inicializace a spuštění jediného servletu aplikace Jenkins CI a tím je servlet Stapler (bližší informace o tomto servletu jsou v kapitole 2.1.4).

Pokud se podaří úspěšně spustit tento servlet, tak je spuštění celé aplikace Jenkins CI z pohledu servlet kontejneru úspěšně provedené a aplikace běží.

Zjištění, které komponenty má servlet kontejner při svém startu zavést, se nachází v souboru `web.xml`, což je standardní konfigurační soubor pro webové aplikace v jazyce Java EE. Může se zde nacházet také konfigurace uživatelských účtů pro přístup k aplikaci a specifikování různých druhů přístupových práv.



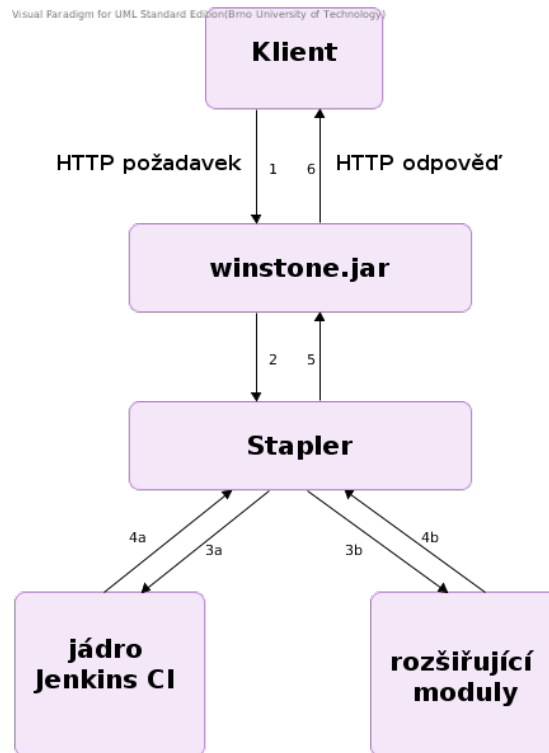
Obrázek 3.2: Průběh spouštění aplikace Jenkins CI při spouštění pomocí vestavěného servlet kontejneru

3.1.2 Průběh komunikace prostřednictvím servlet kontejneru

Po úspěšném zavedení a spuštění aplikace provádí servlet kontejner zpracovávání příchozích požadavků a předává je aplikaci Jenkins CI. Typický zjednodušený průběh komunikace servlet kontejneru s aplikací je znázorněn na obrázku 3.3.

Při komunikaci je příchozí HTTP požadavek zpracován pomocí servlet kontejneru a následně předán příslušnému servletu (dle nastavení pravidel pro směrování požadavků). Jelikož je v aplikaci pouze jediný servlet Stapler, tak je požadavek předán jemu. Tato komponenta následně provede netriviálním způsobem rozhodnutí, které části aplikace předat požadavek k vykonání nebo zda náleží nějakému rozšiřujícímu modulu. Následně stejnou cestou probíhá zaslání HTTP odpovědi zpět klientovi.

Byla zde zachycena pouze jedna z možných činností servlet kontejneru a tou je zpracovávání komunikace protokolem HTTP. Další činnosti kontejneru budou rozebírány později.



Obrázek 3.3: Průběh komunikace aplikace Jenkins CI prostřednictvím vestavěného servlet kontejneru

3.2 Zpětná kompatibilita servlet kontejner v Jenkins CI

V této kapitole jsou blíže rozebírány a analyzovány jednotlivé funkce servlet kontejneru v Jenkins CI. Analýza jeho funkčnosti je zaměřena především na návrh nového servlet kontejneru, který vznikne integrací serveru Undertow. Z této kapitoly vyplynou hlavní požadavky, které bude potřeba řešit při samotné implementaci.

Stávající servlet kontejner je v Jenkins CI již dlouhou dobu a mnoho uživatelů a komponent systému přímo používá jeho specifické parametry, které pocházejí z kontejneru Winstone [3]. Nová implementace tudíž musí zachovat naprosto stejný formát parametrů, aby se výměnou servlet kontejneru nestala část aplikace nebo rozšiřujících modulů nefunkční. Také je potřeba zachovat výchozí hodnoty jednotlivých parametrů.

3.2.1 Funkcionality servlet kontejneru

Při spouštění aplikace je před předáním řízení aplikace servlet kontejneru část parametrů zpracována komponentou `extras-executable-war` (blíže rozebrána v kapitole 3.1) a zbylé parametry jsou přímo předány kontejneru, který podle nich nastaví svou činnost. Kompletní soupis parametrů, které jsou implementovány v servlet kontejneru, je v příloze A. Jednotlivé parametry zde nejsou rozebírány, ale následující analýza se zaměřuje především na klíčové body funkčnosti servlet kontejneru.

Pro zachování zpětné kompatibility musí servlet kontejner v aplikaci Jenkins CI poskytovat následující funkce:

- **Komunikaci nešifrovaným protokolem HTTP**
- **Komunikaci šifrovaným protokolem HTTPS**
- **Komunikaci protokolem AJP13**
- **Umožnit přihlašování a správu přístupových práv**
- **Umožnit restartování a ukončování aplikace přes speciální port**

Protokoly HTTP a HTTPS jsou velmi běžnými protokoly používanými pro komunikaci aplikací ve webovém prostředí a v této práci je očekáváno, že čtenář je s nimi již seznámen. Protokol AJP13 je speciálním typem protokolu, který se využívá především pro efektivnější komunikaci webového serveru a servlet kontejneru [5].

Další činností kontejneru, kterou může vykonávat je správa přístupových práv a umožnění autentizace uživatelů. Konkrétní uživatelské účty mohou být specifikovány při spouštění servlet kontejneru nebo načítány ze standardního konfiguračního souboru webové aplikace `web.xml`.

Poslední z hlavních funkcionalit, které nabízí servlet kontejner v Jenkins CI a je nutné ji zachovat, je možnost ukončit nebo restartovat aplikaci pomocí zaslání speciálního požadavku na zvolený port.

Po integraci serveru Jetty do servlet kontejneru byla přidána ještě možnost komunikace protokolem SPDY, ale tento protokol ještě není v Jenkins CI více využíván a tudíž jeho implementace není zásadní.

3.3 Návrh způsobu integrace serveru Undertow

Na základě analýzy, která proběhla v předchozích kapitolách, je v této kapitole zvolen způsob integrace serveru Undertow do systému Jenkins CI. Obě navržené možné varianty integrace jsou porovnány z různých hledisek a je zvolen způsob, kterým bude následně implementace provedena.

Na závěr kapitoly jsou popsány problémy, které byly objeny při analýze a mají dopad na samotnou integraci a její výsledky.

3.3.1 Varianty integrace

Pro integraci serveru Undertow do systému Jenkins CI jsou možné tyto dva přístupy:

1. Nahrazení pouze serveru Jetty v servlet kontejneru a ponechání zbytku implementace kontejneru Winstone, což by představovalo pouze úpravy části stávajícího kódu.
2. Nahrazení jak serveru Jetty tak kontejneru Winstone. Tento přístup v podstatě znamená provedení celé implementace servlet kontejneru pro Jenkins CI úplně znovu.

Obě varianty integrace mají své klady a zápory. Srovnáme je z hlediska výkonu, proveditelnosti a zpětné kompatibility implementace, abychom mohli následně zvolit vhodnější variantu:

- **Srovnání výkonu:** V první variantě integrace je ponechána stále velmi stará implementace kontejneru Winstone, která zřejmě stále ještě obsahuje nepotřebné součásti

a samotný způsob inicializace není optimalizován pro potřeby serveru Undertow, takže by mohlo docházet ke zbytečnému zpomalování aplikace. Nová implementace může být přímo optimalizována pro potřeby serveru Undertow a poskytovat lepší výkon i kratší dobu spuštění aplikace, která není zanedbatelná při spouštění integračních testů.

- **Srovnání proveditelnosti:** První varianta představuje provedení úprav pouze v částech, kde je přímo integrován server Jetty, zatímco v druhé variantě je potřeba provést znovu celou implementaci integrace servlet kontejneru.

V tomto případě je úprava stávajícího kódu aplikace zřejmě snazším přístupem a také umožňuje provedení rychlejší implementace, protože není nutné navrhovat celý modul, ale pouze jeho části.

- **Srovnání zpětné kompatibility:** Dosažení zpětné kompatibility u první varianty je snazší, jelikož jsou viditelná místa, která je potřeba znovu implementovat a nezanedbat. Nicméně druhá varianta nemá žádné faktické nevýhody k dosažení stejné úrovně zpětné kompatibility jako varianta první.

- **Další hlediska:** Jelikož dlouhodobý vývoj servlet kontejneru v Jenkins CI je na okraji zájmu a jsou prováděny především nejnutnější úpravy, tak celý kód poněkud zdegeneroval a je obtížně čitelný, což je velká závada u open source projektu. Tato skutečnost může bránit dalším přispěvovatelům k provádění potřebných změn.

Z tohoto pohledu by nová implementace mohla přinést mnohem čitelnější způsob řešení a zbavit se zbytečností z původní realizace servlet kontejneru.

Bez ohledu na zvolený způsob integrace bude jeho výsledkem především nový archiv `.jar`, který bude obsahovat příslušnou integraci. Pro jeho využití v systému Jenkins CI je potřeba upravit několik konfigurací a případně upravit jeho načítání při spouštění aplikace v modulu `extras-executable-war`, což bylo rozebíráno v kapitole 3.1.

3.3.2 Zvolení způsobu integrace

Při výběru varianty integrace serveru Undertow do Jenkins CI byly zváženy výhody a nevýhody, které byly popsány v předchozí kapitole. Při nahrazení pouze serveru Jetty by provedení integrace zřejmě probíhalo podstatně snadněji než ve variantě druhé, ale z kvalitativního pohledu se tato varianta jeví jako méně vhodná.

Při implementaci budou tedy nahrazeny obě komponenty stávajícího servlet kontejneru a bude tudíž provedena celá implementace znovu s využitím serveru Undertow. Tato varianta by měla přinést lepší čitelnost zdrojového kódu a poskytnout lepší výkonnost celé aplikace než druhá varianta.

3.3.3 Zjištěné problémy

Při analyzování možností integrace byly zjištěny dva problémy.

Prvním problémem je, že server Undertow je určen pro běh pod virtuálním strojem, který odpovídá specifikaci jazyka Java verze 7, zatímco server Jenkins CI využívá verzi 6. Server Jenkins CI využívá starší verzi z důvodu zpětné kompatibility řešení.

Specifikace jazyka je plně zpětně kompatibilní, takže server Jenkins CI může být spuštěn s novější verzí virtuálního stroje, ale pro okamžité začlenění serveru Undertow do

Jenkins CI je tato skutečnost problém. Pokud by ovšem provedená integrace poskytovala dobré výsledky, tak by bylo možné upravit kód serveru Undertow tak, aby odpovídal starší specifikaci. V těchto verzích jazyka není zásadní rozdíl a úprava by byla jistě možná, ale velmi pracná a stále by potřebovala údržbu při příchodu nových verzí serveru Undertow. Další možností je zvýšit tlak na přechod celé aplikace na novější verzi specifikace jazyka, což výhledově jistě nastane, ale není jisté kdy.

V této práci bude tento problém vyřešen převedením aplikace Jenkins CI na vyšší verzi, což představuje úpravu několika konfigurací.

Druhým problémem je, že při integraci serveru Jetty do Jenkins CI byla přidána možnost využít protokol SPDY, jehož implementace není v současné době v serveru Undertow dostupná. Nicméně tato možnost je zavedena jen krátce a zřejmě ještě není využívána žádnými komponentami, takže by neměl být problém, kdyby nová implementace neobsahovala tuto volbu.

Je důležité dodat, že požadavek na implementaci protokolu SPDY byl v komunitě serveru Undertow vznesen a je také zaznamenán v systému pro sledování chyb¹. Je tedy možné, že tato funkcionality bude v blízké době do serveru Undertow přidána.

¹ Plánovaná implementace protokolu SPDY v serveru Undertow:
<https://issues.jboss.org/browse/UNDERTOW-9>

Kapitola 4

Závěr

Začátek této práce se věnoval seznámení s integračním serverem Jenkins CI, se servery Jetty a Winstone, které jsou součástí jeho servlet kontejneru, a se serverem Undertow, jehož integrace do Jenkins CI je hlavní náplní této práce. V následující části byla analyzována architektura aplikace Jenkins CI a také stav servlet kontejneru, který je v něm integrován.

Po seznámení se s podmínkami pro integraci byly zkoumány možnosti provedení samotné integrace serveru Undertow do Jenkins CI. Byly zkoumány dva způsoby integrace. Jednou možností je nahrazení pouze komponenty Jetty, která vykonává většinu práce v aktuálním kontejneru, zatímco druhou variantou je nahrazení obou součástí. Po důkladném zvážení různých aspektů integrace byla zvolena varianta, kdy budou nahrazeny obě komponenty současného servlet kontejneru a tudíž proběhne zcela nová implementace.

Nebyly objeveny žádné překážky, které by znemožňovaly pokračování v této práci. Dalším krokem v navazujícím diplomovém projektu bude provedení samotné integrace a její důkladné otestování. Po vytvoření funkční implementace bude výsledná aplikace porovnáována s původním řešením z hlediska výkonu. Pokud by byla potvrzena vyšší výkonnost nové implementace, tak by tento přístup mohl nahradit stávající řešení nebo být jeho novou a lepší alternativou.

Literatura

- [1] Kawaguchi, K.: Containers – Jenkins – Jenkins Wiki [online].
<https://wiki.jenkins-ci.org/display/JENKINS/Containers>, 2011-02-03
[cit. 2014-01-06].
- [2] Kawaguchi, K.: Governanace Document – Jenkins – Jenkins Wiki [online].
<https://wiki.jenkins-ci.org/display/JENKINS/Governance+Document>,
2012-03-21 [cit. 2014-01-09].
- [3] Kawaguchi, K.: Winstone is now powered by Jetty [online].
<https://groups.google.com/forum/#!topic/jenkinsci-dev/R1FhPki9z4c>,
2013-10-04 [cit. 2014-01-11].
- [4] Knowles, R.: Winstone servlet container [online].
<http://winstone.sourceforge.net/>, cit. 2014-01-09.
- [5] Milstein, D.: The Apache Tomcat Connector – AJP Protocol Reference [online].
<http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html>, cit. 2014-01-13.
- [6] Prakash, W.: Hudson Web Architecture [online].
<http://hudson-ci.org/docs/HudsonArch-Web.pdf>, 2010 [cit. 2014-01-11].
- [7] Smart, J. F.: *Jenkins: The Definite Guide*. O'Reily Media, Inc., 2011, ISBN
978-1-449-30535-2.
- [8] Wang, R.: What is a Servlet Container? [online].
<http://java.dzone.com/articles/what-servlet-container>, 2013-01-05
[cit. 2014-01-02].
- [9] What is Stapler? [online]. <http://stapler.kohsuke.org/what-is.html>, 2013-11-15
[cit. 2014-01-13].
- [10] Webové stránky projektu Jenkins CI [online]. <http://jenkins-ci.org/>.
- [11] Webové stránky serveru Undertow [online]. <http://undertow.io/>.
- [12] Jetty – Servlet Engine and Http Server [online]. <http://www.eclipse.org/jetty/>,
2014 [cit. 2014-01-02].

Příloha A

Parametry původního servlet kontejneru v Jenkins CI

V této příloze jsou uvedeny parametry, které v systému Jenkins CI poskytuje servlet kontejner. Uvedené parametry jsou přehledem umožňujícím čtenáři detailnější náhlédnutí do možností aktuálního kontejneru, ale nejsou blíže rozebírány a popisovány. Pro věcnou správnost jsou parametry ponechány ve formátu i formulaci jak jsou sepsány v návodu použití a tedy i v původním jazyce, kterým je angličtina.

Seznam možných parametrů je následující:

- `--httpPort` = set the http listening port. -1 to disable, Default is 8080
- `--httpListenAddress` = set the http listening address. Default is all interfaces
- `--httpDoHostnameLookups` = enable host name lookups on incoming http connections (true/false). Default is false
- `--httpKeepAliveTimeout` = how long idle HTTP keep-alive connections are kept around (in ms; default 5000)?
- `--httpsPort` = set the https listening port. -1 to disable, Default is disabled if neither `--httpsCertificate` nor `--httpsKeyStore` are specified, https is run with one-time self-signed certificate.
- `--httpsListenAddress` = set the https listening address. Default is all interfaces
- `--httpsDoHostnameLookups` = enable host name lookups on incoming https connections (true/false). Default is false
- `--httpsKeepAliveTimeout` = how long idle HTTPS keep-alive connections are kept around (in ms; default 5000)?
- `--httpsKeyStore` = the location of the SSL KeyStore file.
- `--httpsKeyStorePassword` = the password for the SSL KeyStore file. Default is null
- `--httpsCertificate` = the location of the PEM-encoded SSL certificate file. (the one that starts with '—BEGIN CERTIFICATE—') must be used with `--httpsPrivateKey`.

- `--httpsPrivateKey` = the location of the PEM-encoded SSL private key. (the one that starts with '—BEGIN RSA PRIVATE KEY—')
- `--httpsKeyManagerType` = the SSL KeyManagerFactory type (eg SunX509, IbmX509). Default is SunX509
- `--spdy` = Enable SPDY. See <http://wiki.eclipse.org/Jetty/Feature/NPN>
- `--ajp13Port` = set the ajp13 listening port. -1 to disable, Default is disabled
- `--ajp13ListenAddress` = set the ajp13 listening address. Default is all interfaces
- `--controlPort` = set the shutdown/control port. -1 to disable, Default disabled
- `--handlerCountStartup` = set the no of worker threads to spawn at startup. Default is 5
- `--handlerCountMax` = set the max no of worker threads to allow. Default is 40
- `--handlerCountMaxIdle` = set the max no of idle worker threads to allow. Default is 5
- `--sessionTimeout` = set the http session timeout value in minutes. Default to what webapp specifies, and then to 60 minutes
- `--mimeTypes=ARG` = define additional MIME type mappings. ARG would be `EXT=MIMETYPE:EXT=MIMETYPE:...` (e.g., `xls=application/vnd.ms-excel:wmf=application/x-msmetafile`)
- `--maxParamCount=N` = set the max number of parameters allowed in a form submission to protect against hash DoS attack (oCERT #2011-003). Default is 10000.
- `--usage / --help` = show this message
- `--version` = show the version and quit
- `--realmClassName` = Set the realm class to use for user authentication. Defaults to `ArgumentsRealm` class
- `--argumentsRealm.passwd.<user>` = Password for user `juserj`. Only valid for the `ArgumentsRealm` realm class
- `--argumentsRealm.roles.<user>` = Roles for user `juserj` (comma separated). Only valid for the `ArgumentsRealm` realm class
- `--fileRealm.configFile` = File containing users/passwds/roles. Only valid for the `FileRealm` realm class
- `--accessLoggerClassName` = Set the access logger class to use for user authentication. Defaults to disabled
- `--simpleAccessLogger.format` = The log format to use. Supports `combined/common/resin/custom` (`SimpleAccessLogger` only)
- `--simpleAccessLogger.file` = The location pattern for the log file(`SimpleAccessLogger` only)